10g rapport

Henrik Flindt
Nicolas Dyhrman
Adrian Joensen

January 2, 2019

Wolf in Moose's Clothing

"A single type of bird can be called a goose, but more is geese, but the plural of moose is not meese, and finally one wolf becomes several wolves. English is like coding. Nobody really know why we do it that way we do, but people on the internet will yell at you for getting it wrong."
-Based on common saying.

The main assignment is the creation of a game, that mimic the relations between mooses and wolfs in a closed park. Each iteration of the game will see the animals move, reproduce and potentially eat a moose. The game is played from the command line interface, by calling its main function called `experimentWAnimals.exe` and give it eight arguments, that fits with the assignment specifications. If done correct, afther the program has run its cause a text file is created with information about the game.

1 Main

The main function was designed to sanitize the user input, and only allowing valid inputs. First it is checked if all the inputs are eighter integers, or in the case of the second argument, a string containing ".txt" in it.
Afterwards several minimum values was implemented, no max values were chosen. While this could be valid as to avoide overflows, all functions works on the premise of list, and thus overflow could only occur if it is an int32 overflow. Since the program begins to take prolonged time when the animal count increases, it is believed that this overflow risk is minimal in a human lifetime. All integer values give (except for the tick int) must be above 0.
While the code can handle 0 animals, a follow effect is that the other values then also should be zero. It was decided to solve this problem with avoiding it. As for the final tick int that can be all the subsets of integers, it is used to check that first round placement (i.e tick = 0) is random, and that a fil is

always created (tick ¡ 0).
The overall design is a row of if-statments. It's a simple code, but quite effective.

2 Manual

fsharpi animalsSmall.fsi animalsSmall.fs animalsSmall.fsx
fsharpc animalsSmall.fsi animalsSmall.fs animalsSmall.fsx
mono animalsSmall.exe animalsSmall.exe animalsSmall.exe

3 Design

We will use Jon's template, so many of the objects will remain the same, except the enviroment to which we will add all the behaviors and events.
To move the animals, we pick a vector from 8('cause there are only 8 directions), check if it's a valid move('cause the animal could be at the edges of the board) and change the animals' position with the vector.
To make the order of animals random, we first look at the wolf list and create a tuple list, where the first element will contain their symbol "w" and the second element will contain the index of the wolves. The same is done to the moose list and we will then join the two lists and shuffle. After the tuple list is shuffled, we will use it to determined who's turn it is.
To shuffle the tuple list, we will use a function with a for loop that will pick a random index, remove the element from the tuple list and also put it in a new list. The function should return the new list as a shuffled tuple list.
To run the simulation, we will use an object with a while loop.

3.1 Two-lists makes a board

3.2 Animals

3.2.1 Moose

3.2.2 Wolf

4 Implementation

```
/// An animal is a base class. It has a position and
    a reproduction counter.
type animal (symb : symbol, repLen : int) =
  let mutable _reproduction = rnd.Next(1,repLen)
  let mutable _pos : position option = None
  let _symbol : symbol = symb

  member this.symbol = _symbol
  member this.position
```

```
    with get () = _pos
    and set aPos = _pos <- aPos
  member this.reproduction = _reproduction
  member this.updateReproduction () =
    _reproduction <- _reproduction - 1
  member this.resetReproduction () =
    _reproduction <- repLen

  override this.ToString () =
    string this.symbol
```

5 White Box Testing

Input	Expected	Result

6 Conclusion